Approximate Fitness Functions

The present invention relates to a method for the evolutionary optimization, to a computer software program for executing such a method as well as to the use of such a method for the optimization of aerodynamic structures.

With reference to figure 1 at first the well-known cycle of an evolutionary algorithm will be explained.

In a step 1 the object parameters to be optimized are encoded in a string called individual. A number of such individuals are randomly generated that comprises the initial parent generation and the quality (fitness) of each individual in the parent generation is evaluated. In a step S2 the parents are reproduced by applying genetic operators called mutation and recombination. Thus, a new generation is reproduced in step S3, which is called the offspring generation. The quality of the offspring individuals is evaluated using a fitness function which is the objective of the optimization in step S4. Finally, depending on the calculated quality value, step S5 selects the best offspring individuals (survival of the fittest) which are used as parents for the next generation cycle if the termination condition in step S6 is not satisfied.

Before evaluating the quality of each individual, decoding may be needed depending the encoding scheme used in the evolutionary algorithm. Note that the steps S2, S3, S4, S5 and S6 are cyclically repeated until the condition for termination of the optimization in step S6 is satisfied.

The algorithm of this evolutionary optimization can be expressed by the following pseudo-code:

```
t:= 0
encode and initialize P(0)
decode and evaluate P(0)
do
        recombine P(t)
        mutate P(t)
        decode P (t)
        evaluate P (t)
        P(t+1) = select P(t)
        encode P (t+1)
        t:=t+1
until terminate
```

Such evolutionary algorithms are known to be robust optimizers that are well suitable for discontinuous and multi-modal objective functions. Therefore, evolutionary algorithms have successfully been applied to mechanical and aerodynamic optimization problems, including preliminary turbine design, turbine blade design, multi-disciplinary rotor blade design, multi-disciplinary wing platform design and a military airframe preliminary design.

Details on the evolutionary algorithms can be found in Dasgupta et al. „Evolutionary Algorithms in Engineering Applications“, Springer-Verlag, 1997 and Miettinnen et al., „Evolutionary Algorithms in Engineering and Computer Science“, John Wiley and Sons. 1999.

One essential difficulty in generation-based approaches to aerodynamic optimization is the huge time consumption due to the high complexity of the aerodynamic analysis and the large number of evaluations needed in the evolutionary optimization. To address this problem, several methods have been developed for constructing approximate models.

One widely used method in design engineering is the Response Surface Methodology, which uses low-order polynomials and the least square estimations.

A more statistically sound method is the Kriging model, which is also called the Design and Analysis of Computer Experiments (DACE) model. In this method, a global polynomial approximation is combined with a local Gaussian process and the Maximum Likelihood is used for parameter estimation. Artificial neural networks, including Multi-layer Perceptrons (MLP) and Radial Basis Function Networks (RBFN) have also been employed to build approximate models for design optimization.

A MLP neural network can be used for building an approximate model. However, it is generally difficult to get a model with sufficient approximation accuracy. One of the problems is the lack of training data for the neural network, because for many real application data collection is a computationally expensive process. This is especially true when the dimension of the problem space is high. Due to this, the approximate model may be of low fidelity and may even introduce false optima. In this case, measures need to be taken to guarantee the correct convergence of the optimization algorithm when approximate models are used.

Managing approximate models in optimization based on evolutionary algorithms has not caught much attention so far. It is known to use a heuristic convergence criterion to determine when the approximate model must be updated. The basic idea is that the convergence of the search process should be stable and therefore, the change of the best solution should not be larger than a user-defined value. An assumption is that the first sets of data points are at least weakly correlated with the global optimum of the original problem, which is not necessarily true for high dimensional systems.

An approach to coupling approximate models with evolutionary algorithm has been proposed in an attempt to balance the concern of optimization with that of design of experiments. The main idea is to maintain the diversity of the individuals and to select those data points that are not redundant for model updating (online learning). In this method, the decision of when to carry out the on-line learning of the approximate model is simply based on a prescribed generation delay.

Most recently it has been proposed to train a neural network with some initial samples to approximate the NK model. During evolution, the fittest individual in the current population is evaluated on the original fitness function for every 50 generations. This individual then replaces the one with the lowest fitness in the training set and the neural network is retrained. It has been found that the evolutionary algorithm becomes misled by the neural network model when the complexity of the original fitness landscape is high. The common weakness in the above methods is that neither the convergence properties of the evolutionary algorithm with approximate fitness functions (correct convergence is assumed) nor the issue of model management is addressed.

In view of the above it is the object of the present invention to reduce the computation time and complexity when managing approximate models for fitness functions. At the same time the convergence properties of the algorithm should essentially not be degraded.

Therefore, a framework for model management with generation based control is proposed. Generally, generation based evolution control is more suitable when the evaluation of the fitness function is implemented in parallel. The main idea is that the frequency at which the original computationally expensive function is called and the approximate model is updated should be determined by the local fidelity of the approximate model. By local fidelity, the fidelity of the model for the region where the current population is located is meant. The lower the model fidelity is, the more frequently the original function should be called and the approximate model should be updated. Since it is hardly possible to build a globally correct approximate model for problems with large dimensionality of the design space, a local model is of more practical importance. With this strategy, the computational cost can be reduced as much as possible while the correct convergence of the evolutionary algorithm can be guaranteed.

The above object is particularly achieved by means of the features of the independent claims. The dependent claims develop further the central idea of the invention.

According to the present invention therefore a method for the evolutionary optimization with approximate fitness functions is proposed. At first an initial design is set up. The initial design is reproduced to create a plurality of offsprings. The quality of the offsprings is evaluated by means of a fitness function. Finally, the offsprings having the highest evaluated quality value are selected for the next generation cycle. Selectively, an original fitness function or an approximate fitness function can be used for the quality evaluation step. According to the invention the frequency of the use of the original fitness function is adaptively adjustable during the optimization process (online adaptation).

The approximate fitness function can be refined (updated) on every use of the original fitness function.

A covariance matrix of the probability density function adapted during the evolutionary optimization can be used for weighting new samples on every update of the approximate fitness function.

The fidelity of the approximate fitness function can be estimated locally on every use of the original fitness function to adjust the future frequency of use of the original fitness function.

On every use of the original fitness functions the current model error can be calculated on the basis of the difference of the result of using the original fitness function and the approximate fitness function, respectively. The future frequency of use of the original fitness function can then be adapted (online) on the basis of the current model error.

The approximate fitness function can be implemented by means of a neural network model.

According to a further aspect of the present invention a computer software program for executing such a method is proposed.

According to the present invention such a method can be used for structural optimization problems, particularly the optimization of aerodynamic bodies, in which case the original function can be calculated by a Navier-Stokes solver.

Further aspects, objects, features and advantages of the present invention will become evident for the man skilled in the art when reading the following detailed description of an embodiment taken in conjunction with the figures of the enclosed drawings.

Figure 1 shows a graphical representation of a generation cycle of an evolution strategy,

Figure 2 is a schematic representation of control cycles using an approximate model or an original function,

Figure 3 shows a flow chart for the online adaptation of the use frequency of an original fitness function,

Figure 4 shows an example of the application of the present invention on a spline encoded blade construction,

Figure 5 shows a simulation of blade optimization with approximate models, and

Figure 6 shows an example for the optimization of blades without approximate models.

Figure 2 shows the technique of calculating a fitness evaluation using both approximate models (for example implemented by means of neural network models) and the original function according to adaptive control cycle. A control cycle is terminated by one fitness evaluation using the original function preceded by no, one or a plurality of fitness evaluations using the approximate model. Obviously the original function is not used for every generation in a control cycle. The control cycle 1 in figure 2 is

composed of two generations using the approximate model and is terminated by one generation using the original function, whereas the control cycle 2 is composed of one generation using the approximate model and is terminated by one generation using the original function. In this example at the end of the control cycle 1 it has been decided to increase the frequency of use of the original function.

Figure 3 shows a flow chart of a process of the present invention for online adaptation of the use frequency $\eta(t)$ of the original function, wherein t is the generation number. After the begin the initial design is fixed in step S1. Then an initial control frequency $\eta(0)$ is fixed in step S7. Depending on the control frequency $\eta(t)$ in an evolution control step S8 it is decided whether the evaluation of the fitness should be performed with the original function (step S9) or with the approximate model (step S11). Usually the evolution control step S8 is designed such that the original function is used in step S9 to terminate a control cycle.

In case the original function is used in step S9 for the evaluation, the data of the result of this evaluation are collected in step S10.

At the end of a generation cycle (composed of steps S2 to S6 as explained with reference to figure 1) it is decided in step S12 whether the current control cycle is finished. If not, the processing goes back in a step S14 to the evolution control decision step S8. If the end of a control cycle is reached, go to step S13 in which the approximate model (the neural network) is updated using the data collected in step S10. Based on an estimation of the model fidelity (as will be explained later on) the future use frequency of the original function $\eta(t)$ is calculated in step S13. Depending on the requirements and the result of the evolution steps S9 or S11, respectively, either the final design is output or the processing goes back in step S14 to the evolution control decision step S8 to start a new control cycle.

The theoretical background of the present invention will now be explained.

The canonical Evolution Strategy (ES) operates on an n-dimensional vector $\vec{x} \in IR^n$. The main operator in ES is the mutation operator. However, unlike in the canonical

genetic algorithm (GA), the mutation operator obtains a direction during search due to the self-adaptation of the strategy parameters. The combination of the genotypes from different parents, carried out by the recombination operator in ES or the crossover operator in GA, usually plays a less significant role or is completely omitted like in this study. The Evolution Strategy is typically combined with a deterministic selection method, either with elitism, i.e., $(\mu+\lambda)$-ES, or without elitism, i.e., (μ,λ)-ES. At the same time, the combination with other selection methods, e.g., the EP tournament selection is known to be successful. The (μ,λ)-ES is applied here, thus μ new individuals are only selected from the λ offspring. The canonical Evolution Strategy (ES) can be described as follows:

$$\vec{x}(t) = \vec{x}(t-1) + \vec{\tilde{z}} \tag{1}$$

$$\sigma_i(t) = \sigma_i(t-1)\exp(r'z)\exp(rz_i), \forall i \in \{1,\ldots,n\} \tag{2}$$

$$\vec{x}, \vec{z} \in IR^n; \qquad z_i, z \approx N(0,1); \qquad \vec{\tilde{z}} \quad N(\vec{0}, \vec{\sigma}(t)^2) \tag{3}$$

where $\vec{x}$ is the parameter vector to be optimized and r, r' and σ_i are the strategy parameters. The values for r and r' are fixed to

$$r = \left(\sqrt{2\sqrt{n}}\right)^{-1}; \qquad r' = \left(\sqrt{2n}\right)^{-1}. \tag{4}$$

The z_i, z and $\vec{\tilde{z}}$ are normally distributed random numbers and a random number vector, respectively, which characterize the mutation exercised on the strategy and the objective parameters. The σ_i are also called step-sizes and are subject to self-adaptation, as shown in equation (2). Since they define the variances σ_i^2 of the normal probability distribution, equation *(3)*, for the mutation in equation (1), they stochastically define a direction and a step-length of the search process in the n-dimensional space. However, arbitrary directions can only be represented by correlated mutations, i.e., the covariance matrix of the normal distribution should

have non-zero off-diagonal elements. Such correlated mutations are realized by the Covariance Matrix Adaptation (CMA) algorithm.

The derandomized Covariance Matrix Adaptation (CMA) differs from the standard ES mainly in three respects:

- In order to reduce the stochastic influence on the self-adaptation, only one stochastic source is used for the adaptation of both objective and strategy parameters. In the derandomized approach the actual step length in the objective parameter space is used to adapt the strategy parameter. Therefore, the self-adaptation of the strategy parameters depends more directly on the local topology of the search space. In the case of the adaptation of one strategy parameter, this can be written as

$$\sigma(t) = \sigma(t-1)\exp\left(\frac{1}{d}\left(|\vec{z}| - E\left[\left|N(\vec{0},\vec{1})\right|\right]\right)\right) \tag{5}$$

$$\vec{x}(t) = \vec{x}(t-1) + \sigma(t)\vec{z} \tag{6}$$

$$\vec{z} \approx N(\vec{0},\vec{1}).$$

Therefore, in the derandomized approach the actual step length in the objective parameter space is used to adapt the strategy parameter. This results in the following, simple, however successful, effect: If the mutation was larger than expected $\left(|\vec{z}| \rangle E\left[\left|N(\vec{0},\vec{1})\right|\right]\right)$, then the strategy parameter is increased. This ensures that if this larger mutation was successful (i.e. the individual was selected), then in the next generation such a larger mutation will again occur, because u(t) was increased. The same argumentation holds if $\left(|\vec{z}| \langle E\left[\left|N(\vec{0},\vec{1})\right|\right]\right)$ The parameter d is used to regulate the adaptation.

- The second aspect is the introduction of the cumulative step size adaptation. Whereas the standard evolution strategy extracts the necessary information for the

adaptation of the strategy parameters from the population (ensemble approach), the cumulative step size adaptation relies on information collected during successive generations (time averaged approach). This leads to a reduction of the necessary population size. The cumulation of steps is termed evolution path by Ostermeier and Hansen and is formally expressed as:

$$\vec{s}(t) = (1-c)\vec{s}(t-1) + c_u\vec{z} \tag{7}$$

The factor c determines the length of the evolution path and the factor $c_u = \sqrt{c(2-c)}$ is needed for normalization, which can be seen by calculating the variance of $\vec{s}(t)$ in the limit $t \to \infty$.

- In the CMA algorithm, the full covariance matrix $\vec{C}$ of the probability density function

$$f(\vec{z}) = \frac{\sqrt{\det(\vec{C}^{-1})}}{(2\pi)^{n/2}} \exp\left(-\frac{1}{2}\left(\vec{z}^T\vec{C}^{-1}\vec{z}\right)\right) \tag{8}$$

is adapted for the mutation of the objective parameter vector. The following description of the CMA algorithm follows the one given in N. Hansen and A. Ostermeier, „Adapting arbitrary normal mutation distributions in evolutions strategies: The covariance matrix adaptation", in Proc. 1996 IEEE Int. Conf. On Evolutionary Computation, pages 312-317, IEEE Press, 1996, details of the implementation can be found in M Kreutz, B. Sendhoff, and Ch. Igel, „EALib: A C++ class library for evolutationary algorithms", Institut für Neuroinformatik, Ruhr-Universität Bochum, 1.4 edition, March 1999.

If the matrix B satisfies $\vec{C} = \vec{B}\vec{B}^T$ and $z_i \approx N(0,1)$, then $\vec{B}\vec{z} \approx N(0,\vec{C})$. The adaptation of the objective vector is given by:

$$\vec{x}(t) = \vec{x}(t-1) + \delta(t-1)\vec{B}(t-1)\vec{z}, \quad z_i \approx N(0,1) \tag{9}$$

where δ is the overall step size. The adaptation of the covariance matrix is implemented in two steps using the cumulative step size approach ($c_{cov} \in (0,1) and c \in (0,1)$) determine the influence of the past during cumulative adaptation).

$$\vec{s}(t) = (1-c)\vec{s}(t-1) + c_u \vec{B}(t-1)\vec{z} \tag{10}$$

$$\vec{C}(t) = (1-c_{cov})\vec{C}(t-1) + c_{cov}\vec{s}(t)\vec{s}^T(t) \tag{11}$$

The next step is to determine the matrix $\vec{B}$ from $\vec{C}$. Since $\vec{C} = \vec{B}\vec{B}^T$ is not sufficient to derive $\vec{B}$, the Eigenvectors of $\vec{C}$ are chosen as column vectors. The reason is that in the last step the overall step size δ has to be adapted. In order to perform this adaptation, the expected length of the cumulative vector $\vec{s}_\delta$ has to be known.

Therefore, the variation vector should be $N(0,\vec{1})$ distributed.

$$\vec{s}_\delta(t) = (1-c)\vec{s}_\delta(t-1) + c_u \vec{B}_\delta(t-1)\vec{z} \tag{12}$$

$$\delta(t) = \delta(t-1)\cdot \exp\left(\beta\left\|\vec{s}_\delta(t)\right\| - \hat{x}_n\right) \tag{13}$$

$\vec{B}_\delta$ equals $\vec{B}$ with normalized columns in order for $\vec{B}_\delta(t-1)\vec{z}$ to be $N(0,\vec{1})$ distributed. $\hat{x}_n$ denotes the expectation of the x_n distribution, which is the distribution of the length of an $N(0,\vec{1})$ distributed random vector.

The CMA algorithm relies on several external parameters which have to be fixed manually.

Parallel Implementation of the Evolutionary Algorithm

The basic motivation to implement an evolutionary algorithm in parallel is to reduce the processing time needed to reach an acceptable solution. This is badly in need when the evaluation of the fitness takes a large amount of time. There are several approaches to parallelization and the one adopted here is the global parallelization. In this approach there is one single population and the evaluation of the individuals is done in parallel. For design problems, the evaluation of the individuals usually takes up the overwhelming part of the total time consujption, therefore, a sub-ilinear speed-up can be achieved if the global parallelization approach is used. The hardware for the implementation of parallel evolutionary algorithms can be very different. In our case, a network of computers with multi-processors is used. The implementation of the parallelization is realized with the Parallel Virtual Machines library.

EMPIRICAL CONVERGENCE STUDIES AND EVOLUTION CONTROL

Convergence of the Evolution Strategy with Neural Networks for Fitness Evaluations

Now the convergence properties of the evolution strategy when a trained multilayer perceptron (MLP) neural network is used for fitness evaluations will be evaluated.

The investigation is carried out on two benchmark problems, the Ackley function and the Rosenbrock function. The Ackley function is a continuous, multimodal test function, which has the following form:

$$f_1(\vec{x}) = -20\exp\left(0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n}x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos 2\pi x_i\right) + 20 + e \qquad (14)$$

where n is the dimension of the function. A neural network is trained to approximate the 2D Ackley function. In the simulation, 600 training data are collected from one run of evolution with the Ackley function as the objective function. This is to simulate the situation that training data are only available from earlier optimization attempts in real world applications. The MLP network used has one hidden layer with 20 nodes.

A false minimum exists near $(x_1,x_2)=(1,-4)$. This can be ascribed to the poor distribution of the training samples. Such samples can be used to simulate the situation frequently encountered in practice, i.e. sparse and poorly distributed training data. Recall that for the discussed problems data collection is expensive. If the evolutionary algorithm is run with this neural network model, the algorithm converges to the false minimum, which is an expected result.

To show that the identified problem is a general one, simulations are also conducted oil the Rosenbrock function.

$$f(\vec{x}) = \sum_{i=1}^{n}\left[10\left(x_i^2 - x_{i+1}\right)^2 + \left(x_i - 1.0\right)^2\right] \tag{15}$$

For his function, the evolutionary algorithm is expected to easily find a near minimum but difficult to locate the global minimum. The output of the fitness function spans from thousands to zero and therefore it becomes a little more difficult for the neural network to learn the mapping. The simulation is conducted on the 2-D Rosenbrock function. Similarly 150 samples are generated from one run of evolution on the 2-D Rosenbrock function, which is much smoother compared to the Ackley function.

Comparing it with the true 2-D Rosenbrock function, it is seen that the left ram of the 2D-Rosenbrock function becomes almost flat due to the poor distribution of the training data. No doubt, evolution with this neural network model is vulnerable to serious errors, because the global minimum of this approximate model lies in the area where the true value is very large. Therefore, an evolutionary optimization based on such an approximate model will converge to the false minimum.

Adding Random Samples

One straightforward idea of dealing with the problem is to add some randomly generated samples for neural network training, first without considering the cost of data generation. In the simulation, 150 random samples are added to the original training

data, both for the 2-D Ackley function and the 2-D Rosenbrock function. It is found that the neural networks are able to learn the main features of the functions and no false minima are present. This is confirmed by running the evolution process with the neural network model as the fitness function, in which a near-optimum is found for both functions.

Unfortunately, adding random samples does not work for high-dimensional systems. To illustrate this, simulations are conducted on the 12-D Ackley function and the 12-D Rosenbrock function. In both cases, 450 randomly generated samples are added to 1000 training data. 450 random samples are added just because in real applications, it is impossible to have as much data as needed. Despite that the neural networks have achieved a good approximation on the training data, incorrect convergence has occurred for both functions.

Improvement of Convergence with Controlled Evolution

In the last subsection, it was shown that using additional training samples is not effective to deal with the problem of "incorrect" convergence of evolutionary algorithms with neural network models. Therefore the concept of evolution control is introduced. Two methods are proposed:

Controlled individuals:
In this approach, part of the individuals (η) in the population (λ in total) are chosen and evaluated with the real fitness function. If the controlled individuals are chosen randomly, it can be called a random strategy. If the best η individuals are chosen as the controlled individuals, it is called a best strategy.

Controlled generations:
In this approach, the whole population of η generations will be evaluated with the real fitness function in every λ generations, where $\eta \leq \lambda$.

Furthermore, when controlled individuals or controlled generations are introduced, new training data are available. Therefore, on-line learning of the neural network will be

applied to improve the approximation of the network model in the region of optimization and in turn to improve the convergence of the evolutionary algorithm.

First the individual based methods for the 12-D Ackley function are investigated. To determine the number of individuals (η) that need to be evaluated using the original fitness function to guarantee correct convergence, an $\eta=1,2,\ldots,11$ is tested for both the random strategy and the best strategy for a $(\mu, \lambda)= (3, 12)$-ES.

All the results are averaged over 10 runs. It can be seen that for the random strategy, convergence of the evolutionary algorithm is not achieved until $\eta\geq 9$. When 5 out of 12 individuals are evaluated with the real fitness function, the reported best fitness is close to the true fitness. When η is larger than 7, a near-optimum is found.

When some of the individuals are evaluated using the real fitness function, new data are available and on-line learning can be implemented. In the following, it is investigated in which way the algorithm can benefit, if these newly available data are used to train the neural network on-line. The evolutionary algorithm reports a correct fitness when only 2 individuals are evaluated with the real fitness function. When about 50% of the whole population are evaluated with the real fitness function, a good near-optimal or optimal solution is found.

Similar results have been obtained for the 12-D Rosenbrock function. It is seen that when $\eta\geq 6$, the algorithm converges to the correct fitness value and a near-optimal solution is found. When on-line learning is introduced, the convergence properties improve considerably and the resulting solution is near-optimal when $\eta\geq 5$. In both cases, the results are based on an average of 10 runs.

It is shown that the best strategy works much better than the random strategy and on-line learning can further improve the convergence properties.

A FRAMEWORK FOR MANAGING APPROXIMATE MODELS

Now a framework for managing approximate models in evolutionary optimization is suggested. Taking into account the fact that parallel evolutionary algorithms are often used for design optimization, the generation-based evolution control strategy is used in this framework.

Generation-based Evolution Control

As explained above, generation-based evolution control is one of the approaches to guarantee the correct convergence of an evolutionary algorithm when the approximate fitness model has false minima. The idea in the generation based evolution control is that in every λ generations there are ηgenerations that will be controlled, where $\eta \leq \lambda$. η should be generally larger than $\lambda/2$ to ensure the correct convergence when false global minima are present in the approximate model. However, an adaptive control frequency (η/λ) is proposed so that the calls of the original expensive fitness function can be reduced as much as possible without affecting the correct convergence of the algorithm.

Determination of Control Frequency

The higher the fidelity of the approximate model is, the more often the fitness evaluation can be made using the model and the smaller η can be. However, it is very difficult to estimate the global fidelity of the approximate model. Thus, a local estimation of the model fidelity is used. This is feasible because the evolution strategy generally proceeds with small steps, i.e., with the normal distribution small mutations are most likely. Therefore the current model error is used to estimate the local fidelity of the approximate model and then to determine the frequency at which the original fitness function is used and the approximate model is updated. The frequency is denoted by η/λ. When λ is fixed, the frequency is solely determined by η.

To get a proper η, heuristic fuzzy rules can be derived as follows:

If the model error is large, then η is large.

Since the rule system has only one input variable, the input-output mapping of such a fuzzy system can be approximately expressed by:

$$\eta(k+1) = \eta_{\min} + \left[\frac{E(k)}{E_{\max}}\right]\left(\eta_{\max} - \eta_{\min}\right), \tag{16}$$

where $[x]$ denotes the largest integer that is smaller than x, η_{max} is the maximal η, $\eta_{max} \leq \lambda$, and η_{min} usually equals 1 so that the information on the model fidelity is always available. E_{max} is the allowed maximal model error and $E(k)$ is the current model error estimation, k denotes the *k-th* cycle of λ generations. Every λ generations is called a *control cycle.*

The estimation of the current model error is carried out before the next control cycle begins. Suppose all the new data in the last η generations are valid (in aerodynamic design, some design may result in unstable fluid dynamics and therefore the data may be invalid) and the population size is P, then there will be ηP data in total. Thus, the model error is estimated as follows:

$$E(k) = \sqrt{\frac{1}{\eta P}\sum_{i=1}^{\eta P}\left(y(i) - y_{NN}(i)\right)^2} \tag{17}$$

where $y(i)$ is the true fitness value and $y_{NN}(i)$ is the fitness calculated using a feed-forward neural network model:

$$y_{NN} = \sum_{j=1}^{H} v_j \theta\left(\sum_{i=1}^{n} w_{ij} x_i\right) \tag{18}$$

where H is the number of hidden nodes, n is the number of inputs, w_{ij} and v_j are the weights in the input layer and output layer, and $\theta(.)$ is the logic function

$$\theta(z) = \frac{1}{1+e^{-z}} \tag{19}$$

The framework for evolutionary optimization with approximate models can be summarized as follows (see also the already explained flow-chart of figure 3):

```
Begin
    Initialize P(0), η(0), λ, E_max, t_max, let k=0
    for t=0 to t_max
        if (t%λ)<η(k)-1
            Use the original fitness function
            Collecting new data
        else
            Use the approximate model
        and if
        if (t%λ)=λ-1
            Estimate the local model fidelity E(k)
            Estimate η(k+1), k=k+1
            Select the data for model updating
            Update the approximate model
        end if
    end for
End
```

Recall that the fidelity of the model is estimated locally based on the error information from the last cycle. Therefore, λ should not be too large.

Weighted Online Learning Using Covariance Matrix

An important issue in on-line learning is how to select new samples for network training to improve the model quality as much as possible. A simple way is to use all the new samples, or a certain number of the most recent data, if the neural network can learn all the data well. Unfortunately, in on-line learning, only a limited number of iterations of training are allowed to reduce the computation time.

There are a number of methods for data selection in neural network training, which are usually called active learning. However, all of these methods rely on sufficient data in order to employ methods from statistics. In the problem outlined in this description, data are sparse and their collection is computationally expensive. At the same time, information about the topology of the search space and even more importantly about the direction of the search process is contained in the covariance matrix which is adapted during the evolutionary process. To exploit this information, the shape of the normal distribution, equation (8), is used to weight the newly generated samples. In this way, the neural network will put more emphasis on the data points that the evolution strategy will most probably visit in the next generation.

Suppose there are N new samples in total, then the cost function for weighted learning is given by

$$E = \frac{1}{N}\sum_{i=1}^{N} p(i)(y(i) - y_{NN}(i))^2 \tag{20}$$

where, $p(i)$ is the weight for sample i

$$p(i) = \exp\left(\frac{1}{2}\vec{x}(i)^T \vec{C}^{-1}\vec{x}(i)\right) \tag{21}$$

The covariance matrix $\vec{C}$ is the one which is adapted during the evolutionary search in the CMA algorithm outlined further above. Before applying the weights to neural network learning, they need to be normalized

$$p(i) = \frac{p(i)}{p_{\max}} \tag{22}$$

where p_{max} is the maximal weight among $p(i)$, $i=1,2,\ldots,N$. In this way, the neural network is able to learn the most important samples, and those with a weight smaller than a given threshold are discarded in learning.

Application of the present invention: Aerodynamic Design Optimization

One of the main difficulties in aerodynamic design optimization is the tremendous time consumption in quality evaluation. Generally, a 2-D or 3-D Navier-Stokes solver with turbulence model is used for computing the fluid dynamics, which usually takes hours of CPU time to get a solution.

Table 1: Generation-based control with weighted online learning

No.	1	2	3	4	5	6	7	8	9	10	Ave rage
Best Fitness	20.6	23.9	22.9	16.7	25.2	18.1	36.0	56.8	38.0	22.5	25.8
Number of calls	996	948	936	900	960	960	876	1164	852	888	948

The objective of the optimization is to maximize the efficiency of a turbine blade and to minimize the bias of the outflow angle from a prescribed value. The maximization of efficiency is realized by the minimization of the pressure loss. In addition, mechanical constraints must be satisfied concerning stability and manufacturing. In order to describe the two dimensional cross section of the airfoil, a spline encoding based on the Non-Uniform Rational B-Splines is used. The spline is constructed from a set on N four-dimensional control points (x, y, z, w) that define the control polygon, where x, y, z are the three dimensional coordinates and w is a weight for the point. A two dimensional cross section is obtained by setting the z-coordinate to zero. To minimize the dimension of the search space the weights of the control points are fixed so that the representation used is a regular B-spline in this example.

Therefore every control point is described only by two parameters, i.e., the x and y coordinate of the point. Figure 4 illustrates a blade (solid line) that is generated by a spline with seven control points. The dotted line shows the corresponding control polygon. The population is initialized with a given blade to reduce the computation

time. Two neural network models are used to approximate the pressure loss and the outflow angle respectively. The development of the pressure loss and the outflow angle during the evolution process are shown in Fig 5. In 185 generations, the Navier-Stokes-Solver has been called 874 times, which corresponds to about 73 generations of evolution on the original fitness function only. Then the evolution has been run for 73 generations and the results are given in Fig. 6. The pressure loss and outflow angle are 0.110 and 69.52 with the approximate models and 0.113 and 69.51 without the approximate models. The final optimization results with approximate models are comparable to those without approximate models, but the evolution process has converged in about 120 generations, in which fewer than 874 calls of Navier-Stokes solver are needed. This achieves an over 30 % reduction of computation time.